The GNU PIES Manual

version 1.2, 11 December 2009

Sergey Poznyakoff.

Published by the Free Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Copyright © 2005, 2006, 2007, 2008, 2009 Sergey Poznyakoff

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being "GNU Pies Manual", and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled "GNU Free Documentation License".

(a) The FSF's Back-Cover Text is: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

Short Contents

1 Introduction 1
2 Inter-Component Dependencies 3
3 Pies Configuration File 5
4 Pies Debugging 33
5 Configuration Examples 35
6 Command Line Usage 37
7 Pies Invocation 41
8 How to Report a Bug 43
A ‘`Inetd.conf`’ Format 45
B User-Group ACLs 49
C GNU Free Documentation License 51
Concept Index 61

Table of Contents

1 Introduction 1

2 Inter-Component Dependencies 3

3 Pies Configuration File 5
- 3.1 Configuration File Syntax 5
  - 3.1.1 Comments 6
  - 3.1.2 Statements 6
  - 3.1.3 Using Preprocessor to Improve the Configuration 8
- 3.2 Component Statement 9
  - 3.2.1 Component Prerequisites 11
  - 3.2.2 Component Privileges 11
  - 3.2.3 Resources 11
  - 3.2.4 Actions Before Startup 13
  - 3.2.5 Exit Actions 13
  - 3.2.6 Output Redirectors 15
  - 3.2.7 Inetd-Style Components 16
    - 3.2.7.1 Built-in Inetd Services 17
    - 3.2.7.2 TCPMUX Services 18
    - 3.2.7.3 Socket Environment Variables 19
    - 3.2.7.4 Exit Actions in Inetd Components 20
  - 3.2.8 Meta1-Style Components 20
  - 3.2.9 Component Syntax Summary 21
- 3.3 Notification 23
- 3.4 Access Control Lists 25
- 3.5 Using `inetd` Configuration Files 27
- 3.6 Using MeTA1 Configuration File 28
- 3.7 Global Configuration 29
- 3.8 Pies Privileges 30
- 3.9 State Files 30

4 Pies Debugging 33

5 Configuration Examples 35
- 5.1 Simplest Case: Using Pies to Run Pmult 35
- 5.2 Using Pies to Run Pmult and MeTA1 35
- 5.3 Running Pies as Inetd 36

6 Command Line Usage 37

7 Pies Invocation **41**

8 How to Report a Bug **43**

Appendix A 'Inetd.conf' Format **45**

Appendix B User-Group ACLs **49**

Appendix C GNU Free Documentation License **51**

Concept Index **61**

1 Introduction

The name `pies` (pronounced '`p-yes`') stands for '`Program Invocation and Execution Supervisor`'. This utility starts and controls execution of external programs, called *components*. Each component is a stand-alone program, which is executed in the foreground. Upon startup, `pies` reads the list of components from its configuration file, starts them, and remains in the background, controlling their execution. When any of the components terminates, `pies` restarts it. Its configuration allows to specify actions other than simple restart, depending on the exit code of the component.

The standard output and standard error streams of a component can be redirected to a file or to an arbitrary `syslog` channel.

This way of operation applies to *init-style* components, called so because of the similarity with the `init` process manager. `Pies` is also able to handle components that receive input on their '`stdin`' and send reply to '`stdout`'. Such components are called *inetd-style* components.

Inetd-style components are not executed right after `pies` startup. Instead, `pies` opens a socket for each of them and listens for connections on these sockets. When a connection arrives, it decides what component the socket corresponds to, and invokes this component to handle that connection. In this case, the connection is bound to component's '`stdin`' and '`stdout`' streams. The '`stderr`' stream can be redirected to a file or to syslog, as described above. This mode of operation is similar to that of the `inetd` utility.

Yet another type of components supported by `pies` are *pass-style* or *meta1-style* components. As the name suggests, this type is designed expressly as a support for MeTA1[1] components, namely `smtps`. This type can be regarded as a mixture of the above two. For each meta1-style component `pies` opens a socket after start-up, and starts the component binary. Once the component is running, `pies` passes it the file descriptor of that socket, through another preconfigured UNIX-style socket. Further handling of the socket is the responsibility of the component itself.

The last flavor of `pies` components are *accept-style* components, which are handled basically as '`inetd-style`' ones, except that after binding to the socket `pies` immediately starts the component, without waiting for an actual connection.

Any number of components of all three styles can be handled simultaneously.

Components are started in the order of their appearance in the configuration file and terminated in reverse order. The same ordering applies when starting or stopping component dependencies,

As an exception, this order is reversed for the components read from MeTA1 configuration files, either included by `include-meta1` statement (see

[1] See http://www.meta1.org

Section 3.6 [include-meta1], page 28) or expressly supplied in the command line (see [config syntax], page 5).

2 Inter-Component Dependencies

A component '`A`' may depend on another components, say '`B`' and '`C`', i.e. require them to be running at the moment of its startup. Components '`B`' and '`C`' are called *prerequisites* for '`A`', while '`A`' is called a *dependency* or *dependent* component of '`B`', '`C`'.

Before restarting any component, `pies` verifies if it is a prerequisite for any other components. If so, it first terminates its dependencies, restarts the component, and then starts its dependencies again, in the order of their appearance in the configuration file.

3 Pies Configuration File

`Pies` reads its settings and component definitions from one or more *configuration files*. The default configuration file is named '`pies.conf`' and is located in the *system configuration directory* (in most cases '`/etc`' or '`/usr/local/etc`', depending on how the package was compiled). This file uses the *native Pies configuration syntax*. Apart from this format, the program also understands configuration files in *inetd* and *meta1* formats.

Alternative configuration files may be specified using '`--config-file`' ('`-c`' command line option), e.g.:

```
pies --config-file filename
```

Any number of such options may be given. The files named in '`--config-file`' options are processed in order of their appearance in the command line. By default, `pies` expects configuration files in its native format. This, however, can be changed by using the '`--syntax=format`' command line option. This option instructs `pies` that any configuration files given after it have are written in the specified *format*. Valid formats are: '`pies`', '`inetd`' and '`meta1`', which stand for pies native format, inetd-style configuration files (see Appendix A [inetd.conf], page 45) and MeTA1-style configuration files. The configuration file format set by the '`--syntax`' option remains in effect for all '`--config-file`' options that follow it, up to the end of the command line or the next occurrence of the '`--syntax`' option. This means that you can instruct `pies` to read several configuration files of various formats in a single command line, e.g.:

```
pies --config-file /etc/pies.conf \
     --syntax=inetd --config-file /etc/inetd.conf \
     --syntax=meta1 --config-file /etc/meta1/meta1.conf
```

The rest of this chapter concerns the `pies` native configuration file format. You can receive a concise summary of all configuration directives any time by running `pies --config-help`. The use of inetd configuration files is covered in Section 3.5 [inetd], page 27 and the use of meta1 configuration files is described in Section 3.6 [include-meta1], page 28

If any errors are encountered in the configuration file, the program reports them on the standard error and exits with status 78.

To test the configuration file without actually starting the server, the '`--lint`' ('`-t`') command line option is provided. It causes `pies` to check its configuration file and exit with status 0 if no errors were detected, and with status 78 otherwise.

Before parsing, configuration file is preprocessed using `m4` (see Section 3.1.3 [Preprocessor], page 8). To see the preprocessed configuration without actually parsing it, use '`-E`' command line option.

3.1 Configuration File Syntax

The configuration file consists of statements and comments.

There are three classes of lexical tokens: keywords, values, and separators. Blanks, tabs, newlines and comments, collectively called *white space* are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent keywords and values.

3.1.1 Comments

Comments may appear anywhere where white space may appear in the configuration file. There are two kinds of comments: single-line and multi-line comments. *Single-line* comments start with '`#`' or '`//`' and continue to the end of the line:

```
# This is a comment
// This too is a comment
```

Multi-line or *C-style* comments start with the two characters '`/*`' (slash, star) and continue until the first occurrence of '`*/`' (star, slash).

Multi-line comments cannot be nested.

3.1.2 Statements

A *simple statement* consists of a keyword and value separated by any amount of whitespace. The statement is terminated with a semicolon ('`;`').

Examples of simple statements are:

```
pidfile /var/run/pies.pid;
source-info yes;
debug 10;
```

A *keyword* begins with a letter and may contain letters, decimal digits, underscores ('`_`') and dashes ('`-`'). Examples of keywords are: '`group`', '`control-file`'.

A *value* can be one of the following:

number
: A number is a sequence of decimal digits.

boolean
: A boolean value is one of the following: '`yes`', '`true`', '`t`' or '`1`', meaning *true*, and '`no`', '`false`', '`nil`', '`0`' meaning *false*.

unquoted string
: An unquoted string may contain letters, digits, and any of the following characters: '`_`', '`-`', '`.`', '`/`', '`:`'.

quoted string
: A quoted string is any sequence of characters enclosed in double-quotes ('`"`'). A backslash appearing within a quoted string introduces an *escape sequence*, which is replaced with a single character according to the following rules:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)
\\	A single backslash ('\')
\"	A double-quote.

Table 3.1: Backslash escapes

In addition, any occurrence of '\' immediately followed by a newline character (ASCII 10) is removed from the string. This allows to split long strings over several physical lines, e.g.:

```
"a long string may be\
 split over several lines"
```

If the character following a backslash is not one of those specified above, the backslash is ignored and a warning is issued.

Two or more adjacent quoted strings are concatenated, which gives another way to split long strings over several lines to improve readability. The following fragment produces the same result as in the example above:

```
"a long string may be"
" split over several lines"
```

Here-document

Here-document is a special construct that allows to introduce strings of text containing embedded newlines.

The `<<`*`word`* construct instructs the parser to read all the following lines up to the line containing only *word*, with possible trailing blanks. Any lines thus read are concatenated together into a single string. For example:

```
<<EOT
A multiline
string
EOT
```

Body of a here-document is interpreted the same way as double-quoted string, unless *word* is preceded by a backslash (e.g. '`<<\EOT`') or enclosed in double-quotes, in which case the text is read as is, without interpretation of escape sequences.

If *word* is prefixed with `-` (a dash), then all leading tab characters are stripped from input lines and the line containing *word*. Furthermore, if `-` is followed by a single space, all leading whitespace is stripped from them. This allows to indent here-documents in a natural fashion. For example:

```
<<- TEXT
    All leading whitespace will be
    ignored when reading these lines.
TEXT
```

It is important that the terminating delimiter be the only token on its line. The only exception to this rule is allowed if a here-document appears as the last element of a statement. In this case a semicolon can be placed on the same line with its terminating delimiter, as in:

```
help-text <<-EOT
        A sample help text.
EOT;
```

list

A *list* is a comma-separated list of values. Lists are delimited by parentheses. The following example shows a statement whose value is a list of strings:

```
dependents (pmult, auth);
```

In any case where a list is appropriate, a single value is allowed without being a member of a list: it is equivalent to a list with a single member. This means that, e.g. '`dependents auth;`' is equivalent to '`dependents (auth);`'.

A *block statement* introduces a logical group of another statements. It consists of a keyword, followed by an optional value, and a sequence of statements enclosed in curly braces, as shown in the example below:

```
component multiplexor {
        command "pmult";
}
```

The closing curly brace may be followed by a semicolon, although this is not required.

3.1.3 Using Preprocessor to Improve the Configuration.

Before parsing, the configuration file is *preprocessed* using external preprocessor `m4`. For a complete user manual, refer to Section "GNU M4" in *GNU M4 macro processor*. In this subsection we assume the reader is sufficiently acquainted with `m4` macro processor.

The external preprocessor is invoked with '`-s`' flag, instructing it to include line synchronization information in its output. This information is then used by the parser to display meaningful diagnostic. An initial set of macro definitions is supplied by the '`pp-setup`' file, located in '*$pre-*

fix`/share/pies/`*version*`/include`' directory (where *version* means the version of the package).

The default '`pp-setup`' file renames all `m4` built-in macro names so they all start with the prefix '`m4_`'. This is similar to GNU m4 '`--prefix-builtin`' options, but has an advantage that it works with non-GNU `m4` implementations as well.

Additional preprocessor symbols may be defined and the existing symbols may be undefined using the following command line options:

'`--define=`*sym*`[=`*value*`]`'
'`-D` *symbol*`[=`*value*`]`'
: Define symbol *sym* as having *value*, or empty, if the *value* is not given.

'`--undefine=`*sym*'
'`-U` *sym*'
: Undefine symbol *sym*.

3.2 Component Statement

`component` [Config]

The `component` statement defines a new component:

```
component tag {
  ...
}
```

The component is identified by its *tag*, which is given as argument to the `component` keyword. Component declarations with the same tags are merged into a single declaration.

The following are the basic statements which are allowed within the `component` block:

`mode` *mode* [Config: component]

Declare the type (style) of the component. Accepted values for *mode* are:

exec
wait
: Define an '`init-style`' component (see [init-style], page 1). This is the default.

inetd
nostartaccept
: Define an '`inetd-style`' component (see [inetd-style], page 1).

pass
pass-fd
: Define a '`meta1-style`' component (see [meta1-style], page 1).

accept
: Define a '`accept-style`' component (see [accept-style], page 1).

`program` *name* [Config: component]
Full file name of the component binary. This binary will be executed (via `/bin/sh -c`) each time `pies` decides it needs to start the component.

To supply command line arguments, use `command` statement.

`command` *string* [Config: component]
Command line for the program. The argument should be just as arguments normally are, starting with the name of the program. The latter may be different from the one specified to `program` statement. Its value will be available to the program as `argv[0]`.

`flags` (*flag-list*) [Config: component]
Define flags for this component. The *flag-list* is a comma-separated list of flags. Valid flags are:

disable
: This component is disabled, i.e. `pies` will parse and remember its settings, but will not start it.

precious
: Mark this component as *precious*. Precious components are never disabled by `pies`, even if they respawn too fast.

wait
: This flag is valid only for '`inetd`' components. It has the same meaning as '`wait`' in '`inetd.conf`' file, i.e. it tells `pies` to wait for the server program to return. See Appendix A [inetd.conf], page 45.

tcpmux
: This is a TCPMUX component. See Section 3.2.7.2 [TCPMUX], page 18.

tcpmuxplus
: This is a TCPMUX+ component. See Section 3.2.7.2 [TCPMUX], page 18.

internal
: This is an internal inetd component. See Section 3.2.7.1 [builtin], page 17.

sockenv
: This inetd component wants socket description variables in its environment. See Section 3.2.7.3 [sockenv], page 19.

resolve
: When used with '`sockenv`', the `LOCALHOST` and `REMOTEHOST` environment variables will contain resolved host names, instead of IP addresses.

`acl { ... }` [Config: component]
Set access control list for this component. See Section 3.4 [ACL], page 25, for a detailed description of access control lists.

The following subsections describe the rest of '`component`' substatements.

3.2.1 Component Prerequisites

Prerequisites (see [component prerequisite], page 3) for a component are declared using the following statement:

`prerequisites` *tag-list* [Config: component]
The argument is either a list of component tags, *defined before this component*, or one of the following words:

all — Declare all components defined so far as prerequisites for this one.

none — No prerequisites. This is the default.

If you wish, you can define dependents, instead of prerequisites:

`dependents` *tag-list* [Config: component]
Declare dependents for this component. *var-list* is a list of component tags.

3.2.2 Component Privileges

The following statements control privileges the component is executed with.

`user` *user-name* [Config: component]
Start component with the UID and GID of this user.

`group` *group-list* [Config: component]
Retain supplementary groups, specified in *group-list*.

`allgroups` *bool* [Config: component]
Retain all supplementary groups of which the user (as given with `user` statement) is a member. This is the default for components specified in '`meta1.conf`' file (see Section 3.6 [include-meta1], page 28).

3.2.3 Resources

`limits` *string* [Config: component]
Impose limits on system resources, as defined by *string*. The argument consists of *commands*, optionally separated by any amount of whitespace. A command is a single command letter followed by a number, that specifies the limit. The command letters are case-insensitive and coincide with those used by the shell `ulimit` utility:

Command	The limit it sets
A	max address space (KB)
C	max core file size (KB)
D	max data size (KB)
F	maximum file size (KB)
M	max locked-in-memory address space (KB)
N	max number of open files
R	max resident set size (KB)
S	max stack size (KB)
T	max CPU time (MIN)
U	max number of processes
P	process priority -20..20 (negative = high priority)

Table 3.2: Limit Command Letters

For example:

```
limits T10 R20 U16 P20
```

Additionally, the command letter '`L`' is recognized. It is reserved for future use ('`number of logins`' limit) and is ignored in version 1.2.

`umask` *`number`* [Config: component]

Set the umask. The *number* must be an octal value not greater than '`777`'. The default umask is inherited at startup.

`env` *`args`* [Config: component]

Set program environment.

Arguments are a whitespace-delimited list of specifiers. The following specifiers are understood:

- (a dash) Clear the environment. This is understood only when used as a first word in *args*.

-*name* Unset the environment variable *name*.

-*name*=*val*
: Unset the environment variable *name* only if its value is *val*.

name Retain the environment variable *name*.

name=*value*
: Define environment variable *name* to have given *value*.

name+=*value*
: Retain variable *name* and append *value* to its existing value. If no such variable is present in the environment, it is created and *value* is assigned to it. However, if *value* begins with a punctuation character, this character is removed from it before the assignment. This is convenient for using this construct with environment variables like `PATH`, e.g.:

```
PATH+=:/sbin
```

In this example, if `PATH` exists, '`:/sbin`' will be appended to it. Otherwise, it will be created and '`/sbin`' will be assigned to it.

name=+*value*
: Retain variable *name* and prepend *value* to its existing value. If no such variable is present in the environment, it is created and *value* is assigned to it. However, if *value* ends with a punctuation character, this character is removed from it before assignment.

`max-instances` *n* [Config: component]
: Sets the maximum number of simultaneously running instances of this component.

3.2.4 Actions Before Startup

The statements described in this subsection specify actions to perform immediately before starting the component:

`chdir` *dir* [Config: component]
: Change to the directory *dir*.

`remove-file` *file-name* [Config: component]
: Remove *file-name*. This is useful, for example, to remove stale UNIX sockets or pid-files, which may otherwise prevent the component from starting normally.

  As of version 1.2 only one `remove-file` may be given.

`pass-fd-timeout` *number* [Config: component]
: Wait *number* of seconds for the '`pass-fd`' socket to become available (see Section 3.2.8 [Meta1-Style Components], page 20). Default is 5 seconds.

3.2.5 Exit Actions

The default behavior of `pies` when an '`init-style`' component terminates is to restart it. Unless the component terminates with 0 exit code, a corresponding error message is issued to the log file. This behavior can be modified using `return-code` statement:

`return-code` [Config: component]

```
return-code codes {
  ...
}
```

The *codes* argument is a list of exit codes or signal names. Exit codes can be specified either as decimal numbers or as symbolic code names from the table below:

Name	Numeric value
EX_OK	0
EX_USAGE	64
EX_DATAERR	65
EX_NOINPUT	66
EX_NOUSER	67
EX_NOHOST	68
EX_UNAVAILABLE	69
EX_SOFTWARE	70
EX_OSERR	71
EX_OSFILE	72
EX_CANTCREAT	73
EX_IOERR	74
EX_TEMPFAIL	75
EX_PROTOCOL	76
EX_NOPERM	77
EX_CONFIG	78

Table 3.3: Standard Exit Codes

Signal numbers can be given either as '`SIG+n`', where *n* is the signal number, or as signal names from the following list: '`SIGHUP`', '`SIGINT`', '`SIGQUIT`', '`SIGILL`', '`SIGTRAP`', '`SIGABRT`', '`SIGIOT`', '`SIGBUS`', '`SIGFPE`', '`SIGKILL`', '`SIGUSR1`', '`SIGSEGV`', '`SIGUSR2`', '`SIGPIPE`', '`SIGALRM`', '`SIGTERM`', '`SIGSTKFLT`', '`SIGCHLD`', '`SIGCONT`', '`SIGSTOP`', '`SIGTSTP`', '`SIGTTIN`', '`SIGTTOU`', '`SIGURG`', '`SIGXCPU`', '`SIGXFSZ`', '`SIGVTALRM`', '`SIGPROF`', '`SIGWINCH`', '`SIGPOLL`', '`SIGIO`', '`SIGPWR`', '`SIGSYS`'.

If the component exits with an exit code listed in *codes* or is terminated on a signal listed in *codes*, `pies` executes actions specified in that '`return-code`' block. The actions are executed in the order of their appearance below:

`exec` *`command`* [Config: return-code]

Execute the supplied external command. Prior to execution, all file descriptors are closed. The *command* inherits the environment from the main `pies` process with the following additional variables:

`PIES_VERSION`
: The `pies` version number (1.2).

`PIES_COMPONENT`
: Tag of the terminated component (see Section 3.2 [Component Statement], page 9).

`PIES_PID` PID of the terminated component.

`PIES_SIGNAL`
: If the component terminated on signal, the number of that signal.

`PIES_STATUS`
: Program exit code.

`action` '`disable | restart`' [Config: return-code]
: If '`restart`' is given, restart the component. This is the default. Otherwise, mark the component as disabled. Component dependents are stopped and marked as disabled as well. Once disabled, the components are never restarted, unless their restart is requested by the administrator.

`notify` *email-string* [Config: return-code]
: Send an email notification to addresses in *email-string*. See Section 3.3 [Notification], page 23, for a detailed discussion of this feature.

`message` *string* [Config: return-code]
: Supply notification message text to use by `notify` statement. See Section 3.3 [Notification], page 23, for a detailed discussion of this feature.

Any number of `return-code` statements are allowed, provided that their *codes* do not intersect.

The `return-code` statements can also be used outside of `component` block. In this case, they supply global actions, i.e. actions applicable to all components. Any `return-code` statements appearing within a `component` block override the global ones.

3.2.6 Output Redirectors

Output redirectors allow to redirect the standard error and/or standard output of a component to a file or `syslog` facility.

`stderr` *type channel* [Config: component]
`stdout` *type channel* [Config: component]
: Redirect standard error (if `stderr`) or standard output (if `stdout`) to the given channel.

The type of redirection is specified by *type* argument:

file
: Redirect to a file. In this case *channel* gives the full name of the file. For example:

```
stderr file /var/log/component/name.err;
```

syslog
: Redirect to a syslog channel. The syslog priority is given by the *channel* argument. Allowed values are: '`emerg`', '`alert`', '`crit`', '`err`', '`warning`', '`notice`', '`info`', '`debug`'. The facility is inherited from the `syslog` statement (see [syslog], page 29), or from the `facility` statement (see below), if given.

Example:

```
stderr syslog err;
```

`facility` *syslog-facility* [Config: component]
Specify the syslog facility to use in syslog redirectors. Allowed *syslog-facility* values are: '`user`', '`daemon`', '`auth`', '`authpriv`', '`mail`', '`cron`', '`local0`' through '`local7`' (all names case-insensitive), or a facility number.

3.2.7 Inetd-Style Components

Inetd-style components are declared using `mode inetd` statement. The '`component`' declaration must contain a '`socket`' statement:

`socket` *url* [Config: component]
Define a socket to listen on. Allowed values for *url* are:

inet[+*proto*]://*ip*:*port*
Listen on IPv4[1]. address *ip* (may be given as a symbolic host name), on port *port*. Optional *proto* defines the protocol to use. It must be a valid protocol name as given in '`/etc/protocols`'. Default is '`tcp`'.

local[+*proto*]://*file*[;*args*]
file[+*proto*]://*file*[;*args*]
unix[+*proto*]://*file*[;*args*]
Listen on the UNIX socket file *file*, which is either an absolute or relative file name, as described above. The *proto* part is as described above. Optional arguments, *args*, control ownership and file mode of *file*. They are a list of assignments, separated by semicolons. The following values are allowed:

user	User name of the socket owner.
group	Owner group of the socket, if it differs from the `user` group.
mode	Socket file mode (octal number between '`0`' and '`777`').
umask	Umask to use when creating the socket (octal number between '`0`' and '`777`').

For example:

```
socket
 "unix:///var/run/socket;user=nobody;group=mail;mode=770";
```

The *file* part may be a relative file name, provided that the `chdir` statement is used for this component (see Section 3.2.4 [Actions Before Startup], page 13).

[1] Support for IPv6 will be added in future versions.

`socket-type` *type* [Config: component]
Sets the socket type. Allowed values for *type* are: '`stream`', '`dgram`', '`raw`', '`rdm`', '`seqpacket`'. Default is '`stream`'. Notice that some socket types may not be implemented by all protocol families, e.g. '`seqpacket`' is not implemented for '`inet`'.

`max-rate` *n* [Config: component]
Specifies the maximum number of times the component can be invoked in one minute; the default is unlimited. A rate of '`0`' stands for '`unlimited`'.

`max-instances` *n* [Config: component]
Sets the maximum number of simultaneously running instances of this component. It is equivalent to the maximum number of simultaneously opened connections.

3.2.7.1 Built-in Inetd Services

Built-in or *internal* services are such inetd-style components that are supported internally by `pies` and do not require external programs. In `pies` version 1.2 those are:

echo	Send back any received data. Defined in RFC 862.
discard	Read the data and discard them. Defined in RFC 863.
time	Return a machine readable date and time as seconds since the Epoch. Defined in RFC 868.
daytime	Return current date and time in human-readable format. Defined in RFC 867.
chargen	Send a continuous stream of ASCII printable characters without regard to the input. Defined in RFC 864
qotd	Send a '`quotation of the day`' text without regard to the input. Defined in RFC 865.
tcpmux	TCP Port Service Multiplexer. Defined in RFC 1078.

A definition of a built-in service component must have the `internal` flag (see [flags], page 10) set. It may not contain `command` or `program` statements, as built-in services do not need external programs. Instead, a *service* declaration must be present:

`service` *name* [Config: component]
Set the built-in service name. Its argument is one of the keywords listed in the above table.

For example, the following component declaration defines a standard TCP-based echo service:

```
component echo {
        socket "inet://0.0.0.0:echo";
        service echo;
        flags internal;
}
```

It corresponds to the following '`inetd.conf`' line:

```
echo stream  tcp     nowait  root    internal
```

Another built-in services are defined in the same manner, replacing '`echo`' in the `service` field with the corresponding service name.

The '`qotd`' service reads the contents of the *qotd file* and sends it back to the client. By default the '`qotd`' file is located in the local state directory and named '*`instance`*`.qotd`' (where *instance* is the name of the `pies` instance; see [instances], page 37). This default location can be changed using the following statement:

`qotd-file` *`file-name`* [Config]
Set the name of the '`quotation-of-the-day`' file.

The text read from the '`qotd`' file is preprocessed, by replacing each LF character (ASCII 10) with two characters: CR (ASCII 13) followed by LF. The resulting text is truncated to 512 characters.

The use of '`tcpmux`' services is covered below.

3.2.7.2 TCPMUX Services

TCPMUX allows to use multiple services on a single well-known TCP port using a service name instead of a well-known number. It is defined in RFC 1078. The protocol operation is as follows. The *master* TCPMUX component listens on a certain TCP port (usually on port 1) for incoming requests. After connecting to the master, the client sends the name of the service it wants, followed by a carriage-return line-feed (CRLF). `Pies` looks up this name in the list of services handled by the master (*subordinate services*) and reports with '`+`' or '`-`' followed by optional text and terminated with the CRLF, depending on whether such service name is found or not. If the reply was '`+`', `pies` then starts the requested component. Otherwise, it closes the connection.

TCPMUX service names are case-insensitive. The special service '`help`' is always defined; it outputs a list of all the subordinate services, one name per line, and closes the connection.

The master TCPMUX service is declared as a usual built-in service, e.g.:

```
component tcpmux-master {
        socket "inet://0.0.0.0:1";
        service tcpmux;
        flags internal;
}
```

Any number of subordinate services may be defined for each master. A subordinate server component definition must contain at least the following statements:

`service` *name* [Config: component]
Sets the name of the subordinate service. The *name* will be compared with the first input line from the client.

`tcpmux-master` *name* [Config: component]
Sets the name of the master TCPMUX service.

`flags` *list* [Config: component]
The `flags` statement (see [flags], page 10) must contain at least one of the following flags:

tcpmux
: A "dedicated" TCPMUX subordinate service. When invoked, it must output the '`+ CRLF`' response itself.

tcpmuxplus
: Simple service. Before starting it, `pies` will send the '`+ CRLF`' reply.

`command` *command-line* [Config: component]
The command line for handling this service.

For example:

```
component scp-to {
        service scp-to;
        flags (tcpmuxplus, sockenv);
        tcpmux-master tcpmux;
        command "/usr/sbin/in.wydawca";
}
```

For TCPMUX services, access control lists are handled in the following order. First, the global ACL is checked. If it rejects the connection, no further checks are done. Then, if the master TCPMUX service has an ACL, that ACL is consulted. If it allows the connection, the subordinate is looked up. If found, its ACL (if any) is consulted. Only if all three ACLs allow the connection, is the service started.

A similar procedure applies for other resources, such as `limits`, `umask`, `env`, `user`, `group`, etc.

3.2.7.3 Socket Environment Variables

If the '`sockenv`' flag is set (see [flags], page 10), the following environment variables are set prior to executing the command:

`PROTO` Protocol name.

`SOCKTYPE` Socket type. See [socket-type], page 17, for a list of possible values.

`LOCALIP` IP address of the server which is handling the connection.

`LOCALPORT`
: Local port number.

`LOCALHOST`
: Host name of the server. This variable is defined only if the '`resolve`' flag is set (see [flags], page 10).

`REMOTEIP` IP address of the remote party (client).

`REMOTEPORT`
: Port number on the remote side.

`REMOTEHOST`
: Host name of the client. This variable is defined only if the '`resolve`' flag is set (see [flags], page 10).

The variables whose names begin with `REMOTE` are defined only for TCP connections.

3.2.7.4 Exit Actions in Inetd Components

Exit actions (see Section 3.2.5 [Exit Actions], page 13) work for '`inet-style`' components. The only difference from '`init-style`' components is that the '`restart`' action is essentially ignored, as it makes no sense to start an '`inet-style`' component without a communication socket.

A common use of `return-code` statement is to invoke an external program upon the termination of a component. For example, the following configuration snippet configures an FTP server and ensures that a special program is invoked after closing each FTP connection:

```
component ftp {
    return-code EX_OK {
        exec "/sbin/sweeper --log";
    }
    mode inetd;
    socket "inet://0.0.0.0:21";
    umask 027;
    program /usr/sbin/in.ftpd
    command "ftpd -ll -C -t180";
}
```

This approach may be used to process `FTP` uploads in real time.

3.2.8 Meta1-Style Components

Meta1-style components are declared using `mode pass` statement. For such components, you must declare both a socket to listen on (see [inetd-socket], page 16 and a UNIX socket name to pass the file descriptor to the component. The latter is defined using `pass-fd-socket` statement:

`pass-fd-socket` *`file-name`* [Config: component]
: The argument is an absolute or relative file name of the socket file. In the latter case, the `chdir` *`dir`* statement must be used for this component

(see Section 3.2.4 [Actions Before Startup], page 13), and the socket will be looked under *dir*.

This socket file is supposed to be created by the component binary upon its startup.

3.2.9 Component Syntax Summary

This subsection summarizes the `component` statements. For each statement, a reference to its detailed description is provided.

```
component tag {
  # Component execution mode.
  # See Section 3.2 [Component Statement], page 9.
  mode 'exec | wait | accept | inetd | nostartaccept | pass-fd | pass';

  # Full name of the program.
  # See Section 3.2 [Component Statement], page 9.
  program name;
  # Command line.
  # See Section 3.2 [Component Statement], page 9.
  command string;

  # List of prerequisites.
  # See Section 3.2.1 [Prerequisites], page 11.
  prerequisites (compnames);
  # List of components for which this one is a prerequisite.
  # See Section 3.2.1 [Prerequisites], page 11.
  dependents (compnames);

  # List of flags.
  # See [flags], page 10.
  flags (flags);

  # Listen on the given url.
  # See Section 3.2.7 [Inetd-Style Components], page 16.
  socket url;

  # Set socket type.
  # See Section 3.2.7 [Inetd-Style Components], page 16.
  socket-type 'stream | dgram | raw | rdm | seqpacket';

  # Service name for built-in inetd component.
  # See Section 3.2.7.1 [builtin], page 17.
  service string;

  # Tag of master TCPMUX component, for subordinate components.
  # See Section 3.2.7.2 [TCPMUX], page 18.
  tcpmux-master string;

  # Pass fd through this socket.
  # See Section 3.2.8 [Meta1-Style Components], page 20.
  pass-fd-socket soket-name;
  # Wait number of seconds for pass-fd socket to become available.
```

```
# See Section 3.2.4 [Actions Before Startup], page 13.
pass-fd-timeout number;

# Maximum number of running instances.
# See Section 3.2.3 [Resources], page 11.
# See Section 3.2.7 [Inetd-Style Components], page 16.
max-instances number;

# Maximum number of times an inetd component can be invoked in
# one minute.
# See Section 3.2.7 [Inetd-Style Components], page 16.
max-rate number;

# ACL for this component.
# See Section 3.4 [ACL], page 25.
acl { ... }

# Override default syslog facility for this component.
facility facility;
# Redirect program's standard output to the given
# file or syslog priority.
# See Section 3.2.6 [Output Redirectors], page 15.
stdout 'file | syslog' channel;
# Redirect program's standard error to the given
# file or syslog priority.
# See Section 3.2.6 [Output Redirectors], page 15.
stderr 'file | syslog' channel;

# Run with this user privileges.
# See Section 3.2.2 [Component Privileges], page 11.
user user-name;
# Retain supplementary group.
# See Section 3.2.2 [Component Privileges], page 11.
group group-name;
# Retain all supplementary groups of which user is a member.
# See Section 3.2.2 [Component Privileges], page 11.
allgroups bool;

# Set system limits.
# See Section 3.2.3 [Resources], page 11.
limits string;

# Force this umask.
# See Section 3.2.3 [Resources], page 11.
umask number;

# Set program environment.
# See Section 3.2.3 [Resources], page 11.
env assignments;

# Change to this directory before executing the component.
# See Section 3.2.4 [Actions Before Startup], page 13.
```

```
    chdir dir;
    # Remove file-name before starting the component.
    # See Section 3.2.4 [Actions Before Startup], page 13.
    remove-file file-name;

    # Actions:
    # See Section 3.2.5 [Exit Actions], page 13.
    return-code exit-code-list {
      # Action to take when a component finishes with this return code.
      action 'disable | restart';
      # Notify these addresses when then component terminates.
      notify email-string;
      # Notification message text (with headers).
      message string;
      # Execute this command.
      exec command
    }
  }
```

3.3 Notification

Pies provides a *notification* mechanism, which can be used to send email messages when components terminate. The exact contents of such notifications and the list of their recipients may depend on the exit code which the component returned. Notification is configured by '`notify`' and '`message`' statements in a '`return-code`' block.

`notify` *email-string* [Config: return-code]

Send email notification to each address from *email-string*. The latter is a comma-separated list of email addresses, e.g.:

```
notify "root@localhost,postmaster@localhost";
```

`message` *string* [Config: return-code]

Supply the email message text to be sent. *String* must be a valid RFC 822 message, i.e. it must begin with message headers, followed by an empty line and the actual message body.

The message may contain variable data in the form of variable references. A *variable* is an entity that holds some data describing the event that occurred. Meta-variables are referenced using the following construct:

```
${name}
```

where *name* is the name of the variable. Before actually sending the message, each occurrence of this construct is removed from the text and replaced by the actual value of the referenced variable. For example, the variables '`component`' and '`retcode`' expand to the name of the exited component and its exit code, correspondingly. Supposing that '`component`' is '`ftpd`' and '`retcode`' is 76, the following fragment:

```
Subject: ${component} exited with code ${retcode}
```

will become:

```
Subject: ftpd exited with code 76
```

The table below lists all available variables and their expansions:

Variable	**Expansion**
canonical-program-name	‘`pies`’
program-name	Program name of the `pies` binary.
package	Package name (‘`Pies`’).
instance	Instance name (see [instances], page 37).
version	Package version (1.2).
component	Name of the terminated component.
termination	Termination cause (see below).
retcode	Component exit code (or signal number, if exited on signal), in decimal.

Table 3.4: Notification Variables

The ‘`termination`’ variable is set so as to facilitate its use with the ‘`retcode`’ variable. Namely, its value is ‘`exited with`’, if the component exited and ‘`terminated on signal`’, if it terminated on a signal. Thus, using

```
${termination} ${retcode}
```

results in a correct English sentence. This message, however, cannot be properly internationalized. This will be fixed in the future versions.

If `message` statement is not given, the following default message is used instead:

```
From: <>
X-Agent: ${canonical-program-name} (${package} ${version})
Subject: Component ${component} ${termination} ${retcode}.
```

Notification messages are sent using an external program, called *mailer*. By default it is `/usr/sbin/sendmail`. You can change it using the following configuration statement:

`mailer-program` *prog* [Config]

Use *prog* as a mailer program. The mailer must meet the following requirements:

1. It must read the message from its standard input.
2. It must treat the non-optional arguments in its command line as recipient addresses.

For example, the following statement instructs `pies` to use `exim` as a mailer:

```
mailer-program /usr/sbin/exim;
```

By default, the mailer program is invoked as follows:

```
/usr/sbin/sendmail -oi -t rcpts
```

where *rcpts* is a whitespace-separated list of addresses supplied in the '`notify`' statement.

The mailer command may be altered using '`mailer-command-line`' statement:

`mailer-command-line` *string* [Config]

Set mailer command line. Notice, that *string* must include the command name as well. The '`mailer-program`' statement supplies the full name of the binary which will be executed, while the first word from the '`mailer-command-line`' argument gives the string it receives as '`argv[0]`'.

The example below shows how to use this statement to alter the envelope sender address:

```
mailer-command-line "sendmail -f root@domain.com -oi -t";
```

3.4 Access Control Lists

Access control lists, or ACLs for short, are lists of permissions that control access to '`inetd`', '`accept`' and '`meta1`'-style components.

An ACL is defined using `acl` block statement:

`acl` [Config]

```
acl {
  definitions
}
```

This statement is allowed both in global context and within a '`component`' block. If both are present, the global-level ACL is consulted first, and if it allows access, the component ACL is consulted. As a result, access is granted only if both lists allow it.

A *named ACL* is an access control list which is assigned its own name. Named ACLs are defined using the '`defacl`' statement:

`defacl` *name* [Config]

```
defacl name {
  definitions
}
```

The *name* parameter specifies a unique name for that ACL. Named ACLs are applied only if referenced from another ACL (either global or a per-component one, or any named ACL, referenced from these). See [acl-ref], page 26, below.

In both forms, the part between the curly braces (denoted by *definitions*), is a list of *access control statements*. There are two types of such statements:

`allow` [*user-group*] *sub-acl host-list* [Config: acl]
`allow` *any* [Config: acl]
Allow access to the component.

`deny` [*user-group*] *sub-acl host-list* [Config: acl]
`deny` *any* [Config: acl]
Deny access to the component.

All parts of an access statement are optional, but at least one of them must be present. The *user-group* part is reserved for future use and is described in more detail in Appendix B [User-Group ACLs], page 49.

The *sub-acl* part, if present, allows to branch to another ACL. The syntax of this part is:

```
acl name
```

where *name* is the name of an ACL defined previously in '`defacl`' statement.

The *host-list* group allows to match client addresses. It consists of the `from` keyword followed by a list of *address specifiers*. Allowed address specifiers are:

addr Matches if the client IP equals *addr*. The latter may be given either as an IP address or as a host name, in which case it will be resolved and the first of its IP addresses will be used.

addr/*netlen*
Matches if first *netlen* bits from the client IP address equal to *addr*. The network mask length, *netlen*, must be an integer number in the range from 0 to 32. The address part, *addr*, is as described above.

addr/*netmask*
The specifier matches if the result of logical AND between the client IP address and *netmask* equals to *addr*. The network mask must be specified in "dotted quad" form, e.g. '`255.255.255.224`'.

filename Matches if connection was received from a UNIX socket *filename*, which must be given as an absolute file name.

The special form '`allow any`' means to allow access unconditionally. Similarly, '`deny any`', denies access unconditionally. Normally, one of these forms appears as the last statement in an ACL definition.

To summarize, the syntax of an access statement is:

```
allow|deny [acl name] [from addr-list]
```

where square brackets denote optional parts.

When an ACL is checked, its entries are tried in turn until one of them matches, or the end of the list is reached. If a matched entry is found, its command verb, `allow` or `deny`, defines the result of the ACL check. If the end of the list is reached, the result is '`allow`', unless explicitly specified otherwise (using the [acl-any], page 26.)

For example, the following ACL allows access for anybody coming from networks '192.168.10.0/24' and '192.168.100.0/24', or any connection that matches the named ACL 'my-nets' (which is defined elsewhere in the configuration file). Access is denied for anybody else:

```
acl {
    allow from (192.168.10.0/24, 192.168.100.0/24);
    allow acl "my-nets";
    deny all;
}
```

3.5 Using inetd Configuration Files

In addition to its native configuration file format, GNU pies is able to read configuration files of several other widely-used utilities. One of these is inetd. The simplest way to use such configuration files is by including them to your main 'pies.conf' using the *include-inetd* statement:

`include-inetd` *file* [Config]

Read components from inetd-style configuration file *file*. The argument may also be a directory, in which case all regular files from that directory are read and parsed as inetd-style configuration files.

The components read from *file* are appended to the pies list of components in order of their appearance.

For example, the following statement reads components from the standard inetd configuration file:

```
include-inetd /etc/inetd.conf;
```

Any number of include-inetd may be specified. For example, the following reads the contents of the '/etc/inetd.conf' configuration file and all files from the '/etc/inetd.d' directory:

```
include-inetd /etc/inetd.conf;
include-inetd /etc/inetd.d;
```

Another way to read inetd configuration files is to supply them in the command line, like this:

```
pies --syntax=inetd --config-file /etc/inetd.conf
```

Notice the '--syntax' option (see [config syntax], page 5). It informs pies that the following files are in inetd format. Of course, several configuration file may be given:

```
pies --syntax=inetd \
     --config-file /etc/inetd.conf --config-file /etc/inetd.d
```

A special option is provided that instructs pies to behave as inetd:

'--inetd' Read configuration from '*sysconfdir*/inetd.conf' and make sure pies state files (see Section 3.9 [State Files], page 30) do not conflict with those from other pies instances.

The GNU Pies package also provides a wrapper that allows to use pies instead of inetd. It is built if the package is configured with the

'`--enable-inetd`' option. The wrapper is then installed in *sbindir* as '`inetd`', possibly replacing the system binary of that name.

The command line usage of the `inetd` wrapper is entirely compatible with that of the usual `inetd` utility, i.e.:

```
inetd [option] [config [config...]] [-- pies-options]
```

Options are:

'`-d`' Increase debug level.

'`-R rate`' Set maximum rate (see [max-rate], page 17).

For convenience, the following additional options are understood:

'`-t`'
'`--lint`' Parse configuration file or files and exit. See [lint], page 5.

'`-s`'
'`--status`'
Display info about the running instance. See [pies-status], page 37.

'`-S`'
'`--stop`' Stop the running instance.

Finally, any additional options `pies` understands may be given to `inetd` after the '`--`' separator.

3.6 Using MeTA1 Configuration File

MeTA1 is a mail transfer agent of new generation, designed to replace Sendmail in the future (`http://www.meta1.org`). It has a modular structure, each module being a component responsible for a particular task. The components are configured in the MeTA1 configuration file '`/etc/meta1/meta1.conf`'.

`Pies` can take a list of components directly from MeTA1 configuration file:

`include-meta1` *file* [Config]
Parse *file* as MeTA1 configuration file and incorporate components defined there into the current component list.

For example:

include-meta1 /etc/meta1/meta1.conf;

Thus, you can use `pies` instead of the default MeTA1 program manager `mcp`. This is particularly useful if you use '`Mailfromd`' (`http://mailfromd.software.gnu.org.ua`) to control the mail flow.

To ensure compatibility with MeTA1, the components read from its configuration file are started in the reverse order (i.e. from last to first), and stopped in the order of their appearance in *file*.

The following `pies` statements are silently applied to all MeTA1 components:

```
allgroups yes;
stderr file compname.log
chdir queue-dir
```

Here, *compname* stands for the name of the component, and *queue-dir* stands for the name of MeTA1 queue directory. The latter is '`/var/spool/meta1`' by default. It can be changed using the following statement:

`meta1-queue-dir` *dir* [Config]
Set name of MeTA1 queue directory.

To override any default settings for a MeTA1 component, add a `command` section with the desired settings after including '`meta1.conf`'. For example, here is how to redirect the standard error of the '`smtps`' component to '`local1.debug`' syslog channel:

```
include-meta1 /etc/meta1/meta1.conf

component smtps {
  facility local1;
  stderr syslog debug;
}
```

3.7 Global Configuration

The statements described in this section affect `pies` behavior as a whole.

`syslog { ... }` [Config]
This block statement configures logging via syslog. It has two substatements:

`tag` *string* [Config: syslog]
Prefix syslog messages with this string. By default, the program name is used.

`facility` *string* [Config: syslog]
Set syslog facility to use. Allowed values are: '`user`', '`daemon`', '`auth`', '`authpriv`', '`mail`', '`cron`', '`local0`' through '`local7`' (case-insensitive), or a facility number.

`umask` *number* [Config]
Set the default umask. The *number* must be an octal value not greater than '777'. The default umask is inherited at startup.

`limits` *arg* [Config]
Set global system limits for all pies components. See Section 3.2.3 [Resources], page 11, for a detailed description of *arg*.

`return-code { ... }` [Config]
Configure global exit actions. See Section 3.2.5 [Exit Actions], page 13, for a detailed description of this statement.

`shutdown-timeout` *`number;`* [Config]
Wait *number* of seconds for all components to shut down. Default is 5 seconds.

3.8 Pies Privileges

Normally, `pies` is run with root privileges. If, however, you found such an implementation for it, that requires another privileges, you may change them using the following three statements:

`user` *`user-name`* [Config]
Start `pies` with the UID and GID of this user.

`group` *`group-list`* [Config]
Retain the supplementary groups, specified in *group-list*.

`allgroups` *`bool`* [Config]
Retain all supplementary groups the user (as given with `user` statement) is a member of.

An example of such implementation is using `pies` to start `jabberd` components: http://www.gnu.org.ua/software/pies/example.php?what=jabberd2.

3.9 State Files

Pies uses several files to keep its state information. The directory which hosts these files is called *state directory*, it is usually '`/var/pies`' or '`/usr/local/var/pies`'). The state directory can be configured at run time:

`state-directory` *`dir`* [Config]
Set the program state directory.

The table below describes the files kept in the state directory. The *instance* in this table stands for the `pies` instance name (see [instances], page 37). Usually, it is '`pies`'.

'*`instance`*`.pid`'
The *PID file*. It keeps the PID number of the running `pies` instance.

'*`instance`*`.ctl`'
The *control file*. This file is used by `pies --restart-component` to pass control requests to the running `pies` instance.

'*`instance`*`.stat`'
The *statistics file*. Used by `pies --status`.

‘*instance*`.qotd`’
: The *Quotation-of-the-day file*. It is used by the ‘`qotd`’ built-in service (see [qotd], page 18).

The following statements allow to redefine state file names. Use them only if the defaults do not suit your needs, and niether the `state-directory` statement nor the ‘`--instance`’ can help:

`pidfile` *`file`* [Config]
: Sets the PID file name.

`control-file` *`file`* [Config]
: Sets the control file name.

`stat-file` *`file`* [Config]
: Sets the statistics file name.

`qotd-file` *`file-name`* [Config]
: Sets the name of the ‘`quotation-of-the-day`’ file.

4 Pies Debugging

The amount of debugging information produced by `pies` is configured by the following statements:

`debug` *`level`* [Config]

Set debugging level. The *level* must be a non-negative decimal integer. In version 1.2 the following debugging levels are used:

1 Log all basic actions: starting and stopping of components, received incoming TCP connections, sending mails. Notify about setting limits. Log pre-startup actions (see Section 3.2.4 [Actions Before Startup], page 13).

2 Log setting particular limits. Log the recomputed alarms.

4 Dump execution environments

6 Debug the parser of MeTA1 configuration grammar.

7 Debug the lexical analyzer of MeTA1 configuration file.

`source-info` *`bool`* [Config]

This statement decides whether debugging messages should contain source information. To enable source information, use:

```
source-info yes;
```

This feature is designed for `pies` developers.

5 Configuration Examples

In this section we provide several examples of working `pies` configuration files.

5.1 Simplest Case: Using Pies to Run Pmult

The example below runs `pmult` (see Section "pmult" in *Mailfromd Manual*) utility with the privileges of '`meta1`' user. Both standard error and standard output are redirected to the syslog facility '`mail`', priorities '`err`' and '`info`', correspondingly.

```
component pmult {
  command "/usr/local/sbin/pmult";
  user meta1s;
  facility mail;
  stderr syslog err;
  stdout syslog info;
}
```

5.2 Using Pies to Run Pmult and MeTA1

The example below is a working configuration file for running `pmult` and all components of MeTA1, configured in '`/etc/meta1/meta1.conf`'. The global `return-code` statement is used to configure `pies` behavior for some exit codes.

```
# Sample pies configuration for running pmult and MeTA1

# Special handling for exit codes that mean the program was
# incorrectly used or misconfigured.
return-code (EX_USAGE, EX_CONFIG) {
  action disable;
  notify "root";
  message <<- EOT
    From: Pies <>
    X-Agent: ${canonical-program-name} (${package} ${version})
    Subject: Component ${component} disabled.

    Component "${component}" has terminated with code ${retcode},
    which means it encountered some configuration problem.
    I will not restart it automatically.  Please fix its configuration
    and restart it manually at your earliest convenience.

    To restart, run ``${program-name} -R ${component}''
    ---
    Wuff-wuff,
    Pies
  EOT;
}

component pmult {
```

```
  command "/usr/local/sbin/pmult";
  user meta1s;
  stderr syslog err;
  stdout syslog info;
}

include-meta1 "/etc/meta1/meta1.conf";
```

5.3 Running Pies as Inetd

This configuration file allows to run `pies` instead of `initd`. It starts two services: '`ftp`' and '`pop3d`', and restricts access to them to two local subnets:

```
acl {
   allow from 10.10.10.0/24;
   allow from 192.168.10.0/27;
   deny from any;
}

debug 3;

component ftp {
   mode inetd;
   socket "inet://0.0.0.0:21";
   umask 027;
   program /usr/sbin/ftpd
   command "ftpd -l -C";
}

component pop3d {
   mode inetd;
   socket "inet://0.0.0.0:110";
   program "/usr/sbin/pop3d";
   command "pop3d --inetd";
}
```

The following is almost equivalent configuration in `inetd` format:

```
ftp  stream tcp  nowait  root /usr/sbin/ftpd  ftpd -l -C
pop3 stream tcp  nowait  root /usr/sbin/pop3d pop3d --inetd
```

This configuration is "almost" equivalent, because the `inetd` format has no way of specifying ACLs and setting the umask.

6 Command Line Usage

When run without arguments, `pies` parses and loads the configuration file, detaches itself from the controlling terminal (becomes a daemon), and starts all components. Before actually starting up, it ensures that no another copy is already running, by looking for a PID file and verifying that the PID listed there is alive and responding. If another copy is running, `pies` refuses to start up.

It is often necessary to run several copies of `pies` with different configuration files. To support such usage, `pies` provides a notion of *instance*. Pies instance is an independent invocation of `pies` that uses a separate configuration file and separate state files (see Section 3.9 [State Files], page 30). Instances are created using the '`--instance`' option:

'`--instance=`*name*'
: Read configuration from '*sysconfdir*`/`*name*`.conf`', use *name* as the base name for state files (i.e., they become '*name*`.pid`', '*name*`.clt`', etc.) and tag all syslog messages with *name*.

For example, the following invocations create three instances of `pies`:

```
pies
pies --instance=inetd
pies --instance=mta
```

The first instance uses the default configuration and state files. The second one reads configuration from '`/etc/inetd.conf`', and the third one reads it from '`/etc/mta.conf`'.

After startup, you can verify the status of the running process using the '`--status`' command line option:

```
$ pies --status
smtps/stderr R  4697
pmult/stderr R  4677
pmult/stdout R  4676
pmult        CR 4678 /usr/local/sbin/pmult
smar         CR 4680 smar -f /etc/meta1/meta1.conf -d 100
qmgr         CR 4691 qmgr -f /etc/meta1/meta1.conf
smtpc        CR 4696 smtpc -f /etc/meta1/meta1.conf
smtps        PR 4698 smtps -d100 -f /etc/meta1/meta1.conf
finger       IL inet+tcp://0.0.0.0:finger /usr/sbin/in.fingerd -u
eklogin      IL inet+tcp://0.0.0.0:eklogin /usr/sbin/klogind -k -c -e
kshell       IL inet+tcp://0.0.0.0:kshell /usr/sbin/kshd -k -c
eklogin      IR 13836  /usr/local/sbin/klogind -k -c -e
```

Each output line contains at least two columns. The first column lists the tag of the component. The second one contains *flags*, describing the type and status of the component. The first flag describes the type:

Flag	**Meaning**
A	Accept-style component
C	Init-style component

E	Command being executed
I	Inetd-style component
P	Pass-style component
R	Output redirector

The second flag is meaningful only for components, i.e. if the first flag is one of '`CAIP`'. Its values are:

Flag	**Meaning**
D	Disabled component
L	Inetd listener
R	Running component
S	Component is stopping
s	Component is sleeping

The next column lists the PID (for running components) or socket address (for Internet listeners), or the string '`N/A`' if neither of the above applies.

If the component is sleeping, the time of its scheduled wake-up is listed in the next column.

The rest of line contains the component command line.

You can restart any component by using the '`--restart-component`' ('`-R`') option, e.g.:

```
$ pies -R pmult smtps
```

To stop all running components and shut down `pies`, use the '`--stop`' ('`-S`') command line option:

```
$ pies --stop
```

Two options are provided for verifying inter-component dependencies. The '`--dump-depmap`' option prints on the standard output the *dependency map*. This map is a square matrix with rows representing dependents and columns representing prerequisites. An '`X`' sign is placed on each crossing which corresponds to the actual dependency. For example:

```
$ pies --dump-depmap
Dependency map:
   0  1  2  3  4
 0
 1
 2     X
 3        X
 4     X  X

Legend:
 0: pmult
 1: smar
 2: qmgr
 3: smtpc
 4: smtps
```

This example corresponds to the configuration file shown in Section 5.2 [Hairy Pies], page 35. To illustrate how to read it, consider the 4th row of the

table. According to the legend, number 4 means ‘`smtps`’ component. There are two ‘`X`’ marks: in columns 1 and 2. This means that ‘`smtps`’ depends on ‘`smar`’ and ‘`qmgr`’.

You can also list prerequisites explicitly:

```
$ pies --dump-prereq
qmgr: smar
smtpc: qmgr
smtps: smar qmgr
```

7 Pies Invocation

This section summarizes `pies` command line options.

'`--config-file=`*file*'
'`-c` *file*' Read configuration from *file*, instead of the default '`/etc/pies.conf`'.

See Chapter 3 [Pies Configuration File], page 5.

'`--config-help`'
Show configuration file summary. See Chapter 3 [Pies Configuration File], page 5.

'`--define=`*sym*`[=`*value*`]`'
'`-D` *symbol*`[=`*value*`]`'
Define symbol *sym* as having *value*, or empty, if the *value* is not given. See Section 3.1.3 [Preprocessor], page 8.

'`--debug=`*level*'
'`-x` *level*'
Set debug verbosity level. See Chapter 4 [Pies Debugging], page 33, for a description of *level*.

'`--dump-depmap`'
Dump dependency map. See [dump-depmap], page 38.

'`--dump-prereq`'
Dump prerequisite charts. See [dump-prereq], page 39.

'`-E`' Preprocess configuration file and exit. See Section 3.1.3 [Preprocessor], page 8.

'`--force`' Force startup even if another instance may be running.

'`--foreground`'
Remain in foreground.

'`--help`' Display a short usage summary and exit.

'`--inetd`'
'`-i`' Run in `inetd`-compatibility mode. It is roughly equivalent to `pies --instance=inetd --syntax=inetd`. See Section 3.5 [inetd], page 27.

'`--instance=`*name*'
Define the name of the `pies` instance. See [instances], page 37.

'`--lint`'
'`-t`'

'`--source-info`'
Show source info with debugging messages. See [source-info], page 33.

‘`--status`’
‘`-s`’ Display info about the running instance. See [pies-status], page 37.

‘`--stderr`’
Log to standard error.

‘`--stop`’
‘`-S`’ Stop the running instance.

‘`--syntax=`*type*’
Define the syntax for parsing the configuration files specified by any ‘`--config-file`’ options that follow this one. Possible values for *type* are:

pies	Native `pies` configuration. See Chapter 3 [Pies Configuration File], page 5.
inetd	‘`Inetd`’-style configuration files. See Appendix A [inetd.conf], page 45.
meta1	‘`meta1`’-style configuration files. See Section 3.6 [include-meta1], page 28.

See [config syntax], page 5, for a detailed description of this option.

‘`--syslog`’
Log to syslog. This is the default.

‘`--rate=`*r*’
Set maximum connection rate (connections per second) for inetd-style components. See [inetd component rate], page 17.

‘`-r`’
‘`--reload`’
‘`--hup`’ Reload the running instance of pies.

‘`-R`’
‘`--restart-component`’
Restart components named in the command line. See [pies-restart], page 38.

‘`--version`’
Display program version and license information and exit.

‘`--undefine=`*sym*’
‘`-U` *sym*’ Undefine symbol *sym*. See Section 3.1.3 [Preprocessor], page 8.

‘`--usage`’ Display a short summary of available options and exit.

8 How to Report a Bug

Send bug-reports and suggestions to bug-pies@gnu.org.ua.

If you think you've found a bug, please be sure to include maximum information needed to reliably reproduce it, or at least to analyze it. The information needed is:

- Version of the package you are using.
- Compilation options used when configuring the package.
- Run-time configuration ('`pies.conf`' file and the command line options used).
- Detailed description of the bug.
- Conditions under which the bug appears.

Appendix A 'Inetd.conf' Format

This appendix describes the format of `inetd` compatible configuration files. See Section 3.5 [inetd], page 27, for the discussion on how to use such files with GNU `pies`.

The `inetd` configuration file has line oriented format. Comments are denoted by a '`#`' at the beginning of a line. Empty lines and comments are ignored. Each non-empty line must be either a service definition, or address specification.

Service definition consists of at least 6 fields separated by any amount of the white space. These fields are described in the following table (optional parts are enclosed in square brackets):

[service-node:]service-name

The service-name entry is the name of a valid service in the file '`/etc/services`'. For built-in services (see Section 3.2.7.1 [builtin], page 17), the service name must be the official name of the service (that is, the first entry in '`/etc/services`'), or a numeric representation thereof. For TCPMUX services, the value of the '`service name`' field consists of the string '`tcpmux`' followed by a slash and the locally-chosen service name (see Section 3.2.7.2 [TCPMUX], page 18). Optionally, a plus sign may be inserted after the slash, indicating that `pies` must issue a '`+`' response before starting this server.

> The '`service-name`' part corresponds to component tag in '`pies.conf`' (see Section 3.2 [Component Statement], page 9). For built-in components, it corresponds to the `service` statement (see Section 3.2.7.1 [builtin], page 17).

Optional '`service-node`' prefix is allowed for internet services. When present, it supplies the local addresses `inetd` should listen on for that service. '`Service-node`' consists of a comma-separated list of addresses. Both symbolic host names and numeric IP addresses are allowed. Symbolic hostnames are looked up in DNS service. If a hostname has multiple address mappings, a socket is created to listen on each address. A special hostname '`*`' stands for `INADDR_ANY`.

socket type

The socket type should be one of '`stream`', '`dgram`', '`raw`', '`rdm`', or '`seqpacket`'. TCPMUX services must use '`stream`'.

> This field corresponds to the `socket-type` statement in '`pies.conf`'. See [socket-type], page 17.

protocol The protocol must be a valid protocol as given in '`/etc/protocols`'. Examples might be '`tcp`' or '`udp`'. TCPMUX services must use '`tcp`'.

> The '`service-node`' prefix and '`socket-type`' field correspond to the `socket` statement in '`pies.conf`'. See [inetd-socket], page 16.
>
> For example, the following line:
>
> ```
> 10.0.0.1:ftp dgram   udp     wait    root  ftpd
> ```
>
> is equivalent to
>
> ```
> socket inet+udp://10.0.0.1:ftp;
> socket-typle dgram;
> ```

wait/nowait[.max-rate]
The '`wait/nowait`' entry specifies whether the invoked component will take over the socket associated with the service access point, and thus whether `pies` should wait for the server to exit before listening for new service requests. Datagram servers must use '`wait`', as they are always invoked with the original datagram socket bound to the specified service address. These servers must read at least one datagram from the socket before exiting. If a datagram server connects to its peer, freeing the socket so that `pies` can go on receiving further messages from the socket, it is said to be a *multi-threaded* server; it should read one datagram from the socket and create a new socket connected to the peer. It should fork, and the parent should then exit to allow `pies` to check for new service requests to spawn new servers. Datagram servers which process all incoming datagrams on a socket and eventually time out are said to be *single-threaded*. Examples of such servers are `comsat` and `talkd`. `tftpd` is an example of a multi-threaded datagram server.

Servers using stream sockets generally are multi-threaded and use the '`nowait`' entry. Connection requests for these services are accepted by `pies`, and the server is given only the newly-accepted socket connected to a client of the service. Most stream-based services and all TCPMUX services operate in this manner. For such services, the invocation rate may be limited by specifying optional '`max-rate`' suffix (a decimal number), e.g.: '`nowait.15`'.

Stream-based servers that use '`wait`' are started with the listening service socket, and must accept at least one connection request before exiting. Such a server would normally accept and process incoming connection requests until a timeout. Datagram

services must use ‘nowait’. The only stream server marked as ‘wait’ is identd (see Section “identd” in *identd manual*).

> The ‘wait’ field corresponds to flags wait in the ‘pies.conf’ file. The ‘nowait’ corresponds to flags nowait. See [flags], page 10.
>
> The ‘max-rate’ suffix corresponds to the max-rate statement. See [max-rate], page 17.

user The user entry contains the name of the user as whom the component should run. This allows for components to be given less permission than root.

> This corresponds to the user statement in ‘pies.conf’. See Section 3.2.2 [Component Privileges], page 11.

program The program entry contains the full file name of the program which is to be executed by pies when a request arrives on its socket. For built-in services, this entry should be ‘internal’.

It is common usage to specify ‘/usr/sbin/tcpd’ in this field.

> This field corresponds to the program statement in ‘pies.conf’. See Section 3.2 [Component Statement], page 9.

server program arguments

The server program arguments should be just as arguments normally are, starting with argv[0], which is the name of the program. For built-in services, this entry must contain the word ‘internal’, or be empty.

> This corresponds to the command statement. See Section 3.2 [Component Statement], page 9.

Address specification is a special statement that declares the ‘service-node’ part (see above) for all the services declared below it. It consists of a host address specifier followed by a colon on a single line, e.g.:

```
127.0.0.1,192.168.0.5:
```

The address specifier from such a line is remembered and used for all further lines lacking an explicit host specifier. It remains in effect until another address specification or end of the configuration is encountered, whichever occurs first.

The following address specification:

```
*:
```

causes any previous default address specifier to be forgotten.

An example of '`inetd.conf`' file with various services follows:

```
ftp             stream  tcp nowait root  /usr/libexec/ftpd    ftpd -l
ntalk           dgram   udp wait   root  /usr/libexec/ntalkd  ntalkd
tcpmux          stream  tcp nowait root  internal
tcpmux/+scp-to  stream  tcp nowait guest /usr/sbin/in.wydawca wydawca
tcpmux/docref   stream  tcp nowait guest /usr/bin/docref      docref
```

Appendix B User-Group ACLs

This appendix describes the '`user-group`' extension for GNU `Pies` ACLs. This extension is reserved for the future use.

The *user-group* ACL statement specifies which users match this entry. Allowed values are the following:

`all` All users.

`authenticated`
: Only authenticated users.

`group` *`group-list`*
: Authenticated users which are members of at least one of groups listed in *group-list*.

For example, the following statement defines an ACL which allows access for any user connected via local UNIX socket '`/tmp/pies.sock`' or coming from a local network '`192.168.10.0/24`'. Any authenticated users are allowed, provided that they are allowed by another ACL '`my-nets`' (which should have been defined before this definition). Users coming from the network '`10.10.0.0/24`' are allowed if they authenticate themselves and are members of groups '`pies`' or '`users`'. Access is denied for anybody else:

```
acl {
    allow all from ("/tmp/pies.sock", "192.168.10.0/24");
    allow authenticated acl "my-nets";
    allow group ("pies", "users") from "10.10.0.0/24";
    deny all;
}
```

Appendix C GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or

to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not

add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such

new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year* *your name*.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘‘GNU Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

> with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

This is a general index of all issues discussed in this manual

-

'-D' ... 9
'-E', introduced ... 5
'-U' ... 9

/

/etc/inetd.conf ... 27
/etc/meta1/meta1.conf ... 28
/etc/protocols ... 16, 46
/etc/services ... 45

A

accept-style components ... 1
access control lists ... 25
acl ... 10, 25
ACL ... 25
ACL in TCPMUX services ... 19
action ... 15
all ... 49
allgroups ... 11, 30
allow ... 26
authenticated ... 49

B

block statement ... 8
boolean value ... 6
builtin services ... 17

C

chargen ... 17
chdir ... 13
command ... 10, 19
Comments in a configuration file ... 6
component ... 1
component ... 9
config-file, --config-file option, introduced ... 5
config-file, --config-file option, summary ... 41
config-help, --config-help option, introduced ... 5
config-help, --config-help option, summary ... 41
configuration file ... 5
configuration file statements ... 6
control-file ... 31

D

daytime ... 17
debug ... 33
debug, --debug option, described ... 33
debug, --debug option, summary ... 41
declaring prerequisites ... 11
defacl ... 25
define, --define option, described ... 9
define, --define option, summary ... 41
deny ... 26
dependencies ... 38
dependency ... 3
dependents ... 3
dependents ... 11
discard ... 17
dump-depmap option, --dump-depmap option option, introduced ... 38
dump-depmap, --dump-depmap option, summary ... 41
dump-prereq, --dump-prereq option, described ... 39
dump-prereq, --dump-prereq option, summary ... 41

E

echo ... 17
env ... 12
escape sequence ... 6
exec ... 14

F

facility ... 16, 29
flags ... 10, 19
force, --force option, summary ... 41
foreground, --foreground option, summary ... 41

G

Global Configuration 29
`group` 11, 30, 49

H

here-document 7
`hup`, `--hup` option, summary 42

I

`include-inetd` 27
`include-meta1` 28
`inetd`, `--inetd` option, described 27
`inetd`, `--inetd` option, summary 41
inetd-style components 1, 16
inetd.conf 27
init-style components 1
`instance`, `--instance` option, summary 41
`internal` flag 17
internal services 17

L

`limits` 11, 29
`lint`, `--lint` option, introduced 5
`lint`, `--lint` option, summary 41
list 8

M

m4 8
mailer 24
`mailer-command-line` 25
`mailer-program` 24
`max-instances` 13, 17
`max-rate` 17
`message` 15, 23
`meta1-queue-dir` 29
meta1-style components 1, 20
`mode` 9
multi-line comments 6

N

Notification 23
`notify` 15, 23

P

`pass-fd-socket` 20
`pass-fd-timeout` 13
pass-style components 1
`pidfile` 31
`pies.conf` 5
`pp-setup` 8
preprocessor 8
prerequisite 3
`prerequisites` 11
prerequisites, declaring 11
privileges 11, 30
`program` 10

Q

qotd 17
`qotd-file` 18, 31
quoted string 6

R

`rate`, `--rate` option, summary 42
`reload`, `--reload` option, summary ... 42
`remove-file` 13
repeater 15
`restart-component`, `--restart-component` option, described 38
`restart-component`, `--restart-component` option, summary 42
`return-code` 13, 30

S

`sendmail` 24
`service` 17, 19
`shutdown-timeout` 30
simple statements 6
single-line comments 6
smtps 1
`sockenv` 19
`socket` 16
socket environment variables 19
`socket-type` 17
`source-info` 33
`source-info`, `--source-info` option, summary 41
`stat-file` 31
state files 30

`state-directory` 30
statement, block 8
statement, simple 6
statements, configuration file 6
`status`, `--status` option, summary ... 41
`stderr` 15
`stderr`, `--stderr` option, summary ... 42
`stdout` 15
`stop`, `--stop` option, described 38
`stop`, `--stop` option, summary 42
string, quoted 6
string, unquoted 6
`syntax`, `--syntax` option, summary ... 42
`syslog` 29
`syslog`, `--syslog` option, summary ... 42

T

`tag` 29
tcpmux 17
TCPMUX 18
`tcpmux-master` 19
time 17

U

`umask` 12, 29
`undefine`, `--undefine` option, ... 42
`undefine`, `--undefine` option, described 9
`user` 11, 30